An Open-Source Platform for High-Performance Non-Coherent On-Chip Communication

Thomas Benz <tbenz@iis.ee.ethz.ch>
Motivation

- **Trend towards more complex ICs**
  - larger die sizes
  - feature scaling (Intel 20A)
  - Increasing heterogeneity (ML accelerators)

- **Huge amount of high BW memory**
  - on-chip: SRAM
  - off-chip: HBM2E

➢ Need for high-BW point-to-point data transfers
Major on-chip protocols

- Intel: Ultra Path Interconnect
- AMD: Scalable Data Fabric
- IBM: Power9 on-chip interconnect
- ARM: Advanced eXtensible Interface (AXI) (and others)

Not available for third parties (or only under royalties)

open standard that can be used without royalties
AXI Implementations

- Synopsys: DesignWare IP Solutions for AMBA Interconnect
- Cadence: VIP only
- ARM: AMBA Products (CoreLink NIC-400, CCI-500, ...)
- Xilinx: LogiCORE IP Products (Interconnect, Data Width Converter, ...)

- FOSS, technology-independent implementation?

proprietary, expensive

licensed with Xilinx products, but FPGA only generated & adapted with IP Integrator
ETH Zurich PULP platform AXI

- FOS, technology-independent
- AXI4 and AXI4-Lite synthesizable IPs in SystemVerilog
  - Written and optimized by hand
- Extensive verification infrastructure
  - UVM-compatible
- Full architectural description and extensive documentation
- Fully customizable and extensible
  - User signals are routed
  - Achieve best performance by customizing to the application
AXI Architecture / Terminology

- 5 independent transaction channels
  - Valid, ready, last handshake
- Components
  - Master, slave, interconnect
- Master initiates AXI operation to slave
  - Set of required op. -> transaction
  - Burst of individual data beats
AXI Multiplexer

- Connect multiple slave ports to one master port

- **Operation:**
  - Multiplexing forward channel
  - Fair round-robin arbitration
  - Demultiplexing backward channel

- **Complexity: backward channel**
  - Critical path: $O(\log S)$ (arbitration)
  - Area: $O(S)$ (arbitration)
AXI Demultiplexer

- Connects one slave port to multiple master ports

- **Operation:**
  - Externally select master port
  - Store id information to route reordered responses

- **Complexity:** keep ordering
  - Critical path: $O(M), O(I)$
  - Area: $O(M), O(2^I)$
AXI Crossbar (X-bar)

- Connects N master ports to M slave ports

- Operation:
  - Address decoding (slave ports)
  - Master selection (demultiplexer)
  - Multiplexing
  - Optionally: add cuts, error slave

- Complexity:
  - Critical path: $O(M + I)$ (demux)
  - Area: $O(MS + 2^I S)$ (S demux, M mux)
Additional Design IPs

- ID remapper and serializer
- Data Upsizer and downsizer
- Simplex and duplex on-chip SRAM controller
- AXI-attached last level cache (LLC)
- Multi-channel AXI DMA engine
- And many more 😊
Building large Systems from our IPs

- Our IPs are written and optimized by hand in SystemVerilog

- Naturally: use SystemVerilog to create the AXI system
  - AXI has many signals -> Tedious and error-prone process
  - We provide a macro-based solution to create AXI types and connect buses
    - Even with SV generate constructs: limit fast exploration

- One solution: Use HLS to describe topology (and generate V code)
  - We would have to throw away our optimized IPs 😞

- Solution: use template-based HLS strategy
Solder: Template & IP-based HLS

- **Python-based application**
  - Configuration file for:
    - Parameters, Addresses
    - Other user-defined constants
  - Mako templateing for SV base

- **Address maps, routes propagated and sanity checks performed**

- **Interconnect, SoC, testbench, documentation, linker script, ... generation**

- **Generates understandable (modifiable) SystemVerilog**
Example: AXI Adaptation

**JSON config:**
- wide data width: 512
- narrow data width: 64

---

**solder.py**

```
from soldertools.solderlib import *

config = {'config': {'debug': False, 'tool': 'solder', 'validator': 'validator', 'input': 'example.json', 'output': 'example.sv', 'format': 'systemverilog'}, 'json': {'config': {'debug': False, 'tool': 'solder', 'validator': 'validator', 'input': 'example.json', 'output': 'example.sv', 'format': 'systemverilog'}, 'json': {'config': {'debug': False, 'tool': 'solder', 'validator': 'validator', 'input': 'example.json', 'output': 'example.sv', 'format': 'systemverilog'}}}
```

---

**Generated SystemVerilog Code**

```systemverilog
// Wide to Narrow Crossbar //
//
.wide data width: 512
.narrow data width: 64

//soc wide xbar.out_soc.narrow

.scl_wide xbar.in_soc.wide, "soc wide narrow iwc"

...
Example: AXI X-Bar

JSON config:
- wide data width: 512
- narrow data width: 64
- cache base addr: ...
- cache size: ...

Configuration of fixed values

```
{{module}}

// Address map of the 'wide xbar quadrant sl' crossbar.
xbar_rule 48 t [4:0] wideXbarQuadrantsSlAddrmap;
assign wideXbarQuadrantsSlAddrmap = {
    '{ t idx: 1, start addr: 40'h1000000000, end addr: 40'h2000000000 },
    '{ t idx: 2, start addr: cluster base addr[0], end addr: cluster base addr[0] + ClusterAddressSpace },
    '{ t idx: 3, start addr: cluster base addr[1], end addr: cluster base addr[1] + ClusterAddressSpace },
    '{ t idx: 5, start addr: cluster base addr[3], end addr: cluster base addr[3] + ClusterAddressSpace }
};
wide xbar quadrant sl in req t [4:0] wide xbar quadrant sl in req;
wide xbar quadrant sl in resp t [4:0] wide xbar quadrant sl in rsp;
wide xbar quadrant sl out req t [4:0] wide xbar quadrant sl out req;
wide xbar quadrant sl out resp t [4:0] wide xbar quadrant sl out rsp;

axi_xbar #(
    .tfq(WideXbarQuadrantsSlTfg),
    .Connectivity(30'b1111111111111111111111111111111),
    .Ato ostream (0),
    .slv ax chan t(axi a48 d512 14 u0 w chan t),
    .slv ax chan t(axi a48 d512 17 u0 w chan t),
    .w chan t (axi a48 d512 14 u0 w chan t),
    .slv b chan t (axi a48 d512 34 u0 b chan t),
    .slv b chan t (axi a48 d512 17 u0 b chan t),
    .w chan t (axi a48 d512 34 u0 w chan t),
    .slv ar chan t(axi a48 d512 14 u0 ar chan t),
    .slv ar chan t(axi a48 d512 34 u0 ar chan t),
    .slv r chan t(axi a48 d512 34 u0 r chan t),
    .slv r chan t(axi a48 d512 17 u0 r chan t),
    .w chan t (axi a48 d512 34 u0 w chan t),
    .resp t (axi a48 d512 14 u0 resp t),
    .resp t (axi a48 d512 17 u0 resp t),
    .rule t (xbar_rule 48 t)
) i wide xbar quadrant sl1
```

solder.py

Generated SystemVerilog Code
Future Development: Fast Exploration

- Solder generates the Interconnect from
  - Config file and SystemVerilog template

- It has the full overview of the instantiated AXI IPs and their configuration

- From fitted models it is possible:
  - Estimate the critical path of each IP
  - Estimate the size of each IP

- Estimate timing and area of full AXI system
  - Do ultra fast (automated) exploration
Future Development: AXI Extensions

- **Error correction**
  - Space-grade applications
  - Use redundant data (e.g. parity)
  - User Signals
  - Create / check integrity at source / sink

- **Memory stream-based extensions**
  - Custom burst types
  - Encode more complex memory streams
    - N-D transfers with regular strides
    - Scatter / gather operations with arbitrary memory streams
Conclusion

- We created a synthesizable FOS AXI4 implementation, that can compete with industry-grade solutions
  - Check it out at https://github.com/pulp-platform/axi

- Our AXI4 implementation is written in SystemVerilog, optimized by hand and fully characterized

- We have a template-based HLS approach to create SoCs and their AXI4 interconnects
  - Check out our reference RISCV system: https://github.com/pulp-platform/snitch